VŠB – Technická universita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Klientská a ovládací část programu Netcon

Client and Control Part of Program Netcon

2011 Bc. Zdeněk Ryška

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum: Podpis: _____

Zdeněk Ryška

Poděkování

Chtěl bych poděkovat vedoucímu své diplomové práce panu inženýru Davidu Seidlovi za cenné rady a připomínky, které měly velmi pozitivní vliv nejen na kvalitu vyvíjeného systému, ale i této práce.

Abstrakt

Tato práce se věnuje tvorbě klientů a webové ovládací části systému Netcon, který byl jako bakalářská práce tvořen studentem Janem Bajerem. Jan Bajer pracuje na tvorbě serverové části a já se v této práci věnuji klientům pro platformu Windows a Linux a také webovému ovládacímu rozhraní.

Klíčová slova:

OpenSSL, C++, Visual Studio, soket, C#, ASP.NET, JAVA

Abstract

This work is focused to creating client and web control part of system Netcon, which was formed as an undergraduate thesis by student Jan Bajer. Jan Bajer is working on creating the server component and I'm focused to clients for Windows and Linux platform and web-based interface.

Keywords:

OpenSSL, C++, Visual Studio, socket, C#, ASP.NET, JAVA

Obsah

Úvod

Tato diplomová práce je zaměřená na tvorbu klientských aplikací pro systém Netcon, jehož serverovou část tvoří Bc. Jan Bajer. Hlavním cílem tohoto systému je snížení nákladů na elektrickou energii, jejíž ceny se neustále zvyšují a také usnadnění práce správcům sítí.

Dnes se velmi často stává, že hlavně ve školách, nebo větších organizacích běží počítače často přes noc, případně v několikahodinových pauzách mezi výukou. Pokud má daná organizace pouze jednoho případně dva správce tak ani není v jejich silách několikrát denně kontrolovat vizuálně, zda jsou opravdu všechny počítače vypnuté. Tento problém velmi efektivně řeší systém Netcon. Je jím možné počítače na dálku vypnout, zapnout a zkontrolovat jejich stav - velmi rychle a jednoduše.

V první části této diplomové práce bude popsán komunikační protokol systému a použité technologie.

Během tvorby byl systém přejmenován na Bcontrol (Binary control system), z důvodu shody jmen původního názvu s jiným komerčním projektem a také proto, že se jedná o jiný systém nežli Netcon. Tedy v textu bude pod pojmem Netcon zamýšlen původní systém a Bcotrol bude systém aktuálně vyvíjený.

Po popisu komunikačního protokolu a architektury systému, se bude tato práce věnovat tvorbě klientských aplikací pro platformy Linux a Windows, kde budou popsány rozdíly obou verzí. Následně bude popsána tvorba grafických ovládacích aplikací pro oba systémy. A také tvorba instalačních balíků a to systémem NSIS pro operační systém Windows a formou .deb balíku pro operační systém Linux.

Poslední část této diplomové práce bude věnována tvorbě webového ovládacího rozhraní, kterým bude možné systém pohodlně ovládat.

Výstupem této diplomové práce by měly být plně funkční klientské aplikace pro systémy Linux a Windows a také webové ovládací rozhraní spolupracující se serverem Bc. Jana Bajera.

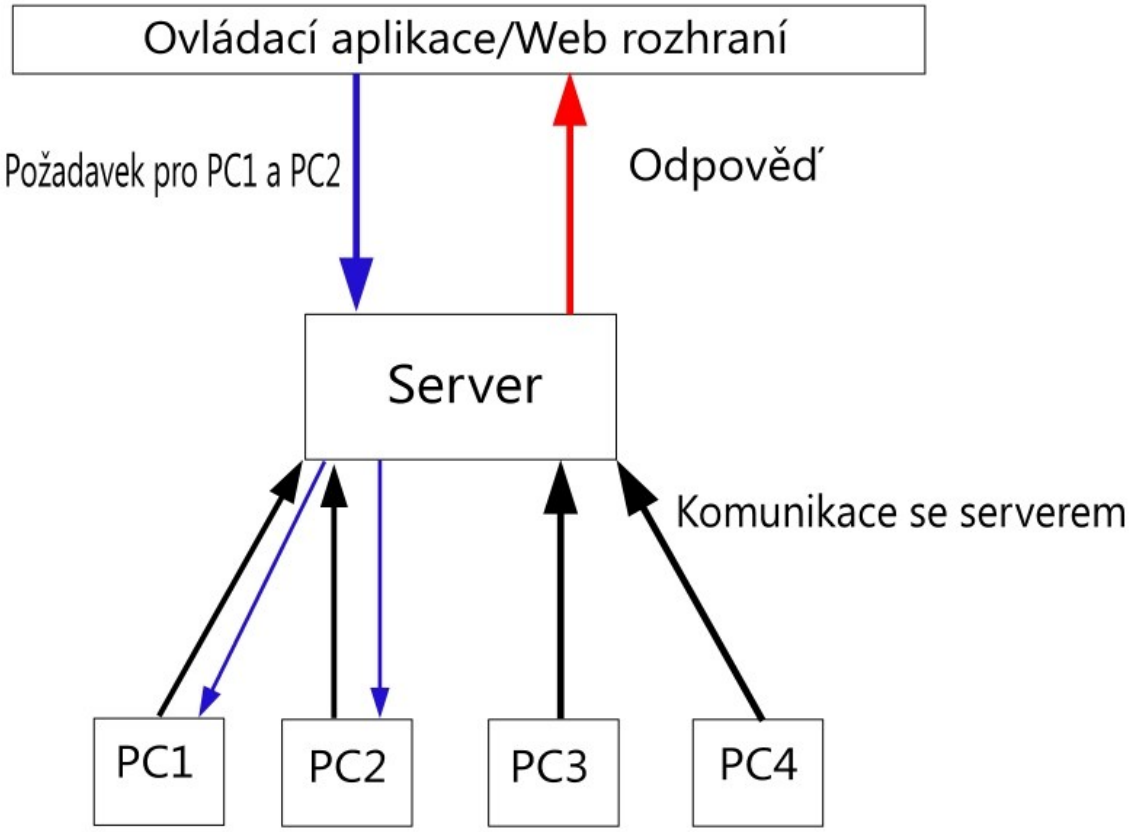
1. Protokol Systému

Systém používá pro komunikaci mezi klienty a serverem ustálený protokol, který bude v této kapitole popsán.

Nejprve bude popsána komunikace serveru s klienty a poté komunikace klientů se serverem.

1.1 Komunikace Server – Klient

Zprávy zasílané ovládací aplikací jsou ve tvaru – „operace|ip adresa|ip adresa…“



obr. č. 1 Náhled komunikace

1.1.1 Druhy podporovaných operací

- **vypnout** – oznámí žádost o vypnutí počítače uživateli a obdrží časový limit, za který se má klient opět připojit k serveru pro ověření stavu. Možnost snížení

časového limitu zde slouží k rychlejšímu zjištění, zda uživatel vypnutí odložil nebo se počítač vypnul

- **vynutitVypnout** - vypne počítač bez upozornění uživatele. Slouží k vynucenému vypnutí počítače v případě, že nějaký program brání vypnutí nebo uživatel vypnutí neustále odkládá
- **Restartovat** - oznámí žádost o restartování počítače uživateli a obdrží časový limit, za který se má klient opět připojit k serveru pro ověření stavu. Odeslání času má zde stejné opodstatnění jako u vypnutí
- **vynutitRestartovat** – restartuje počítač bez upozornění uživatele. Opět obdobné jako u operace vynutitVypnout
- **nic|čas** – tedy například „nic|60“ oznámí klientovi, že pro něj nemá žádnou operaci a čas, který má čekat do opětovného připojení na server. Časová hodnota se udává ve vteřinách.

1.2 Komunikace Klient – Server

Klient posílá serveru po každém připojení informaci o svém stavu.

Prihlaseno(Linux) ; Prihlaseno(Windows) ; v případě, že je uživatel přihlášen v systému

SYSTEM(Linux) ; SYSTEM(Windows) ; v případě že je počítač na přihlašovací obrazovce

Tedy výše popsaná komunikace jsou jediná data, která probíhají mezi serverem a klienty. Jelikož server slouží pouze jako prostředník mezi klientem a ovládacím programem či webovým rozhraním, tak mu lze poslat prakticky libovolný řetězec. Jen bude nutné v ovládacím programu, nebo webovém rozhraní upravit formátování řetězce, případně jeho zpracování pro korektní zobrazení.

1.3 Použité knihovny

Systém používá OpenSSL knihovny pro šifrování komunikace mezi serverem, klienty a ovládací aplikací. Knihovny byly použity ve verzi 1.0.0c což byla poslední stabilní verze v době kompilace. Oproti předchozím verzím obsahuje hlavně opravy bezpečnostních chyb a jedná se o doporučovaný update.[6]

```
o Fix for security issue CVE-2010-4180
o Fix for CVE-2010-4252
o Fix mishandling of absent EC point format extension.
o Fix various platform compilation issues.
o Corrected fix for security issue CVE-2010-3864.
```

Konfigurační soubory jsou ve verzi pro Windows i Linux tvořeny pro knihovnu LibConfuse ve verzi 2.7. V případě Windows verze byly knihovny staticky přilinkovány k programu. Verze 2.7 nepřinesla žádné nové funkce - jsou v ní obsaženy pouze opravy chyb.

Knihovny jsou nutné v systému Windows přidat do VC++ Directories programu Visual Studio 2008, v systému Linux jsou linkovány kompilátorem. Postup přiložení knihoven bude popsán v popisu tvorby klientů.

2. Architektura Systému

Systém v předchozí verzi fungoval na principu klient-sever-klient, kde byl klient na koncovém počítači pasivní, a server byl aktivní. Tedy klient komunikoval se serverem pouze na jeho podnět. Tento princip kladl značné nároky na stabilitu serverové části. Server musel v jednom okamžiku vytvořit stejný počet vláken jako kontaktovaných počítačů. To znamená, že při požadavku na vypnutí 200 počítačů server musel okamžitě vytvořit 200 vláken a po jejich obsloužení je zase odstranit. Tato nárazová zátěž vedla k občasné nestabilitě serverové části.

Dalším problémem mohla být architektura sítě – server by se nemusel dostat na všechny pracovní stanice přes NAT.

A v neposlední řadě také velmi vysoká odezva systému – server kontaktoval klienty a postupně dostával odpovědi, které odesílal ovládacímu programu. Tato operace trvala dle počtu kontaktovaných pracovních stanic i několik desítek vteřin. Což rozhodně nepřispělo k příjemné práci se systémem.

Tyto problémy jsou odstraněny v této nové verzi systému. Nyní je klient aktivní a v pravidelných intervalech kontaktuje server a posílá mu informaci o sobě. Jediným požadavkem na síťovou infrastrukturu je server dostupný z celé sítě, což není velký problém zajistit. Funkce „info“, která byla obsažena v předchozí verzi systému, se nyní od klientů nevyžaduje, ale zpracuje ji přímo server tak, že odešle uložená data od klientů ovládací aplikaci. Server si pamatuje IP adresu počítače a jeho stav, který se s každým připojením klienta aktualizuje. Pokud serveru přijde požadavek na určitou operaci, uloží ji pod ip adresou počítače – tedy zamění standardně odesílanou odpověď „nic|čas“ za „operace|čas“, kterou obdrží klient po připojení k serveru a provede požadovanou operaci.

V předchozí verzi byla nutná znalost pouze cílových IP adres, které měl server kontaktovat a u klientské verze se nemuselo nastavovat nic – nyní musí být v konfiguračním souboru klientů uvedena IP adresa serveru, aby bylo možné komunikaci realizovat. Pokud je zadaná IP adresa chybná, nebo server neběží, není možné žádným způsobem klienty kontaktovat.

Dále přibyla možnost nešifrované komunikace v systému a u komunikace šifrované pomocí OpenSSL je nyní možné nastavit hloubku ověřování certifikátů. V případě deaktivování OpenSSL šifrování jsou data šifrována vnitřní šifrou (bitovým posunem), aby se po síti nepřenášel čistý text (plain text). Tato možnost byla přidána pro případnou nestabilitu, nebo nemožnost nasazení OpenSSL šifrování. Případné dešifrování dat posílaných po síti nepředstavuje bezpečnostní hrozbu a není ani důvod, aby se o tento krok někdo pokoušel.

Dalším vylepšením je již zmíněný konfigurační soubor, který značně usnadňuje a zpřehledňuje nastavení aplikace. Jeho možnosti budou dále detailněji popsány.

3. Klient pro operační systém Linux

3.1 Výběr programovacího jazyka

V případě výběru programovacího jazyka pro systém Linux bylo zvoleno spolehlivé a hlavně rychlé C++. Klient běží na cílové stanici jako služba, v případě Linuxových systémů jako „Daemon“ a není žádoucí, aby zatěžoval systém, což se i povedlo dodržet. Ke snadnějšímu rozhodování přispělo i to, že zdrojové kódy bylo možné z velké části použít i v operačním systému Windows. Změny, které bylo nutné provést, budou popsány v části věnované tvorbě klientské aplikace pro platformu Windows. Nabízelo se i použití #ifdef bloků, ale z důvodu využití Visual C++ šablony pro službu v systému Windows bylo nutné od této myšlenky upustit.

Tedy pro samotné jádro služby byl zvolen jazyk C++ a dále bylo nutné volit jazyk, ve kterém bude napsána grafická nadstavba programu pro zajištění komunikace s uživatelem a následnému poskytnutí možností interakce, která je žádoucí pro možnost odložení vypnutí, nebo restartování systému. V tomto případě byl zvolen programovací jazyk Java, který je velmi dobře použitelný pod Linuxovým systémem, nabízí dostatečné možnosti a Java Runtime Environment nemá příliš velkou velikost pro instalaci, nehledě na to, že na většině linuxových distribucí bývá často předinstalovaný.

Vývojové prostředí pro C++ část aplikace bylo zvoleno Netbeans, které nabízí kontrolu syntaxe a přehledně formátuje zdrojový kód programu. Kompilace, ale probíhala klasicky v terminálu pomocí makefile souboru. Pro tvorbu grafické nadstavby v programovacím jazyce JAVA bylo opět použito vývojové prostředí Netbeans.

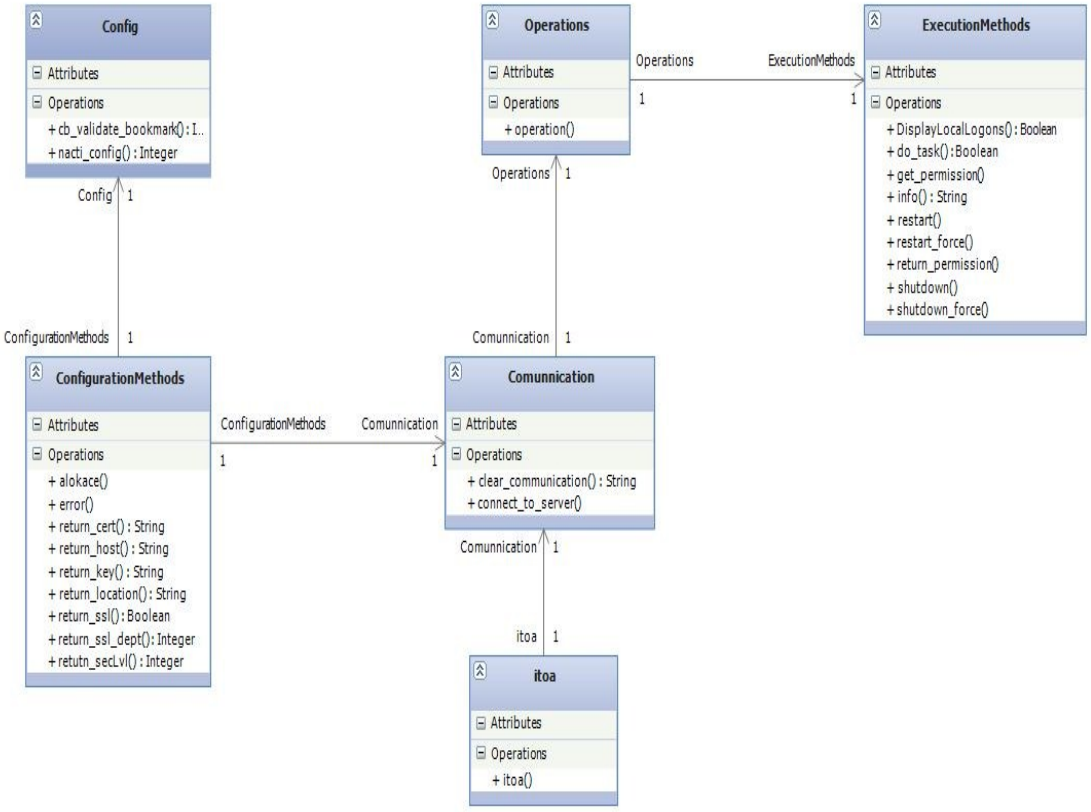
3.2 Struktura programu

Program je rozdělen na několik tříd, které reprezentují určitou funkčnost. Nejdůležitější je třída communication.cpp, která se stará o navázání spojení se serverem a celou komunikaci. V hlavičkovém souboru functions.h jsou sdruženy veškeré použité hlavičkové soubory a v souboru classes.h jsou definovány jednotlivé třídy, vazby mezi nimi, jejich metody a atributy.

Communication.cpp	Navazuje komunikaci se serverem (šifrovanou či nešifrovanou) a dále dle přijatých dat se rozhoduje o provedení patřičné akce.
Configuration.cpp	Třída, která umožňuje načtení proměnných z konfiguračního souboru. Načítá struktury a typy daných proměnných a také validuje načítaný konfigurační soubor. Nakonec kontroluje, zda

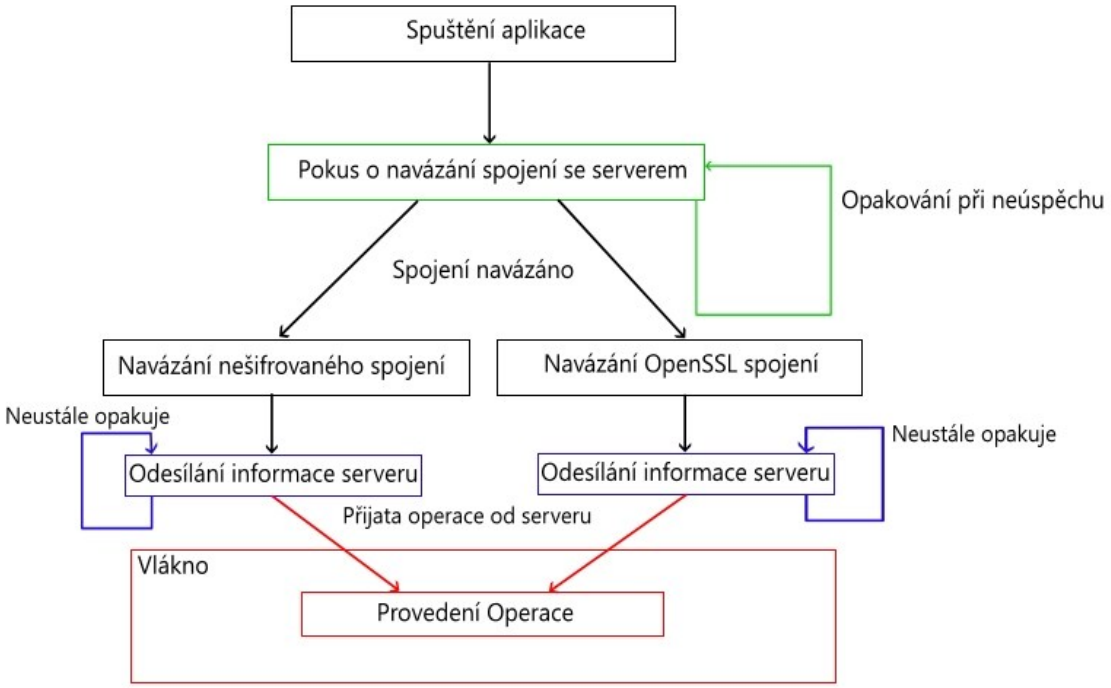
	jsou odkazované OpenSSL certifikáty skutečně obsaženy v cílových adresářích.
Configuration_methods.cpp	Tato třída obsahuje metody, které umožní načtení konkrétních proměnných z konfiguračního programu.
Execution_methods.cpp	V této třídě jsou obsaženy metody, které jsou volány ze třídy operations.cpp. Obsahuje metody jako vypnout, restartovat..
operations.cpp	Tato třída v podmínkou if rozhoduje o operaci, která bude provedena na základě dat přijatých od serveru.
Itoa.cpp	Třída sloužící k převodu datového typu int na datový typ string. Jedná se o převzaté hotové řešení.

3.3 Nákres struktury programu



Obr. č. 2 Náhled na strukturu programu

3.4 Znázornění běhu programu



Obr. č. 3 Náhled na funkci programu

3.5 Spuštění programu

Při startu programu se dle nastavení proměnné „debug“ rozhodne, zda se program bude spouštět jako daemon, nebo jako klasická konzolová aplikace. Tento mód je přítomen pro možnost umístění ladících výpisů a usnadnění hledání případných chyb. Standardně je debug režim vypnut a je provedena „deamonizace“, při které se vytvoří PID soubor a veškeré výstupy programu se zapisují do systémového log souboru umístěného v adresáři „/var/log/“ s názvem daemon.log.

Poté se program pokusí připojit na IP adresu serveru načtenou z konfiguračního souboru. V případě neúspěšného připojení, tyto pokusy neustále opakuje v deseti-vteřinovém intervalu.
Po úspěšném navázání spojení k serveru je načtena proměnná určující druh komunikace – šifrovaná/nešifrovaná a dle hodnoty této proměnné pokračuje běh programu.

V případě šifrované komunikace jsou načteny umístění certifikátů z konfiguračního souboru a klient naváže se serverem šifrované spojení. Od této chvíle je veškerá komunikace probíhající mezi serverem a klientem zabezpečená pomocí OpenSSL. Nyní klient v cyklu odesílá serveru

informaci o stanici, na které běží. Časový limit klient obdrží od serveru – celý běh programu je velmi dobře patný na výše uvedeném obrázku č. 3.

Analogicky je to v případě nastavení nešifrovaného spojení, kde se místo zabezpečené OpenSSL komunikace, použije interní šifra realizovaná bitovým posunem. Tedy data mezi klientem a serverem nejsou, jak by se mohlo z názvu zdát, nešifrovaná, ale jsou upravená jednoduchou šifrou.

Klient tedy v cyklu odesílá data o stanici, na které běží šifrovaná pomocí OpenSSL, nebo bitovým posunem, do té doby dokud nejsou přijata od serveru data s názvem operace (názvy operací a jejich vysvětlení je popsáno v části věnované protokolu) k provedení. V tom případě je vytvořeno nové vlákno, kterému je jako parametr zadán obdržený název operace. Klient potom dle stavu stanice (Přihlášen/Nepřihlášen uživatel), spustí grafickou nadstavbu s určitým parametrem, nebo přímo operaci provede.

Po provedení této akce klient pokračuje v činnosti (odesílání informací o stanici) dokud není restartován, nebo vypnut. Tento průběh je shodný i u klientské služby pro operační systém Windows.

3.6 Tvorba klientské části pro systém Linux

Tato část práce bude zaměřena na samotnou tvorbu klientské aplikace pro systém Linux. Budou zde ukázky kódu částí samotné služby a poté bude popsána tvorba grafické nadstavby, která byla tvořena v programovacím jazyce JAVA.

Z této části by mělo být na úrovni kódu pochopitelné, jak program pracuje.

3.6.1 Struktura tříd

Základní třídy a jejich funkčnost byla již popsána výše. Nyní se zaměříme na popis použitých metod v daných třídách a jejich účel.

Communication.cpp

- `string clear_communication(string data)` – tato metoda zajišťuje nešifrovanou komunikaci se serverem. Jako parametr očekává data, která se mají serveru odeslat a vrací data od serveru přijatá.

- `void thread_function(void* data)` – slouží pro potřeby vytvoření nového vlákna v případě přijetí požadavku na provedení operace od serveru.
- `void error(string msg)` – tato metoda slouží k zápisu případných chybových stavů do logu systému Linux. Použitý je standartní daemon.log umístěný v adresáři /var/log/.
- `Void connect_to_server(Configuration &obj_sett)` – jedná se o hlavní metodu, která navazuje a realizuje komunikaci se serverem. V této metodě se rozhoduje o použití šifrované/nešifrované komunikace.

Configuration.cpp

- `int cb_validate_bookmark(cfg_t *cfg, cfg_opt_t *opt)` – tato metoda slouží k validaci sekcí konfiguračního souboru.
- `int load_config()` – obsahuje inicializace struktur a proměnných, které mohou být načteny. Také kontroluje, zda zadaná souborová cesta skutečně vede k daným certifikátům.

Operations.cpp

- `void operation(string oper, pthread_mutex_t semafor_id)` – metoda dostane jako parametr řetězec s názvem operace, který byl přijat od serveru a rozhodne v bloku `if` o provedení příslušné akce, která se volá ze třídy Execution_methods.cpp.

Execution_Methods.cpp

- `string info()` – tato metoda se provádí před každým připojení k serveru, aby se mohli odeslat získaná data.
- `void reboot()` – metoda provede restartování počítače s možností odložení v případě přihlášeného uživatele (JAVA aplikace), opačném případě počítač ihned restartuje zavoláním funkce shutdown s parametrem –r.

- `void reboot_force()` – metoda provede vynucené restartování počítače. Bez jakéhokoli upozornění případného uživatele.
- `void shutdown()` – metoda provede vypnutí počítače s možností uživatelského odložení. (JAVA aplikace) V případě že není nikdo přihlášen počítač přímo vypne systémovou funkcí `halt`.
- `void shutdown_force()` – metoda provede vynucené vypnutí počítače. Bez jakéhokoli upozornění případného uživatele.

Configuration_Methods.cpp

- `void alokace()`
- `string return_host()` – vrací ip adresu serveru načtenou z konfiguračního souboru.
- `int return_port()` – vrací číslo portu načtené z konfiguračního souboru. Je nutné nastavit různé porty pro šifrovanou a nešifrovanou komunikaci (8889/8888).
- `bool return_ssl()` – nastavení této proměnné rozhodne o použití šifrovaného OpenSSL spojení se serverem.
- `string return_cert()` – vrací řetězec reprezentující cestu k certifikačnímu souboru.
- `string return_key()` – vrací řetězec reprezentující cestu k privátnímu klíči klienta.
- `string return_location()` – vrací řetězec reprezentující cestu ke klíči serveru.
- `int return_ssl_deptCheck()` – výstupem této funkce je číselná hodnota, která reprezentuje hloubku ověřování certifikátů.
- `int return_secLevel()` – výstupem je číselná hodnota reprezentující typ zabezpečení OpenSSL komunikace.

3.7 Vysvětlení kódu

Základním je v případě klientské části programu soubor main.cpp, který má na starosti spuštění aplikace. Program nejprve započne daemonizaci, při které jsou veškeré výstupy aplikace uzavřeny a vytvořen PID soubor. Poté je vytvořena instance třídy Configuration.cpp pro možnost načtení dat s konfiguračního souboru.

Pro práci s konfiguračním souborem není jazyk C++ nijak vybaven, proto byla použita velmi povedená konfigurační knihovna libConfuse[7]. Samotná libConfuse je knihovna určená k analýze a vybrání hodnot z konfiguračního souboru. Dříve se jmenovala libcfg, ale pro

odlišení od jiných knihoven s podobným názvem, byla přejmenována na libConfuse. Knihovna je napsaná v jazyce C a soustředí se hlavně na jednoduché použití a velmi snadnou integraci do kódu libovolného projektu. Podporuje prakticky jakkoli dlouhé sekce hodnot různých datových typů (strings, integers, boolean…).

Program dále pokračuje vytvořením instance třídy Communication.cpp, kde se zavolá metoda connect_to_server() a jako parametr se jí předá objekt, přes který se načítají proměnné z konfiguračního souboru (adresa serveru, port, nastavení šifrování…).
V této metodě jsou nejdříve (volitelně) inicializovány OpenSSL knihovny pokud je nastaveno jejich použití v konfiguračním programu.

```
if(obj_sett.vrat_ssl() == true)
{
   SSL_library_init();
   SSL_load_error_strings();
   SSLeay_add_ssl_algorithms();
}
```

Dále metoda pokračuje nastavením soketu pro komunikaci se serverem. Parametr AF_UNSPEC nastaví nespecifikovanou adresní verzi (vybere se dle potřeby IPv4, nebo IPv6), SOCK_STREM zajišťuje spolehlivé oboustranné spojení, které využívá přenos dat typu OOB a parametr IPPROTO_TCP, který volí jako přenosový protokol TCP.

Následně probíhá inicializace paměti pro převod načteného čísla portu *int* z konfiguračního souboru na datový typ *char*, který bude použit ve volání funkce getaddrinfo. Pro převod je použita třída itoa.cpp, která byla převzata (viz. komentáře v kódu).

```
vysl = itoa(obj_sett.return_port(), vysl, 10);
```

Takto převedené číslo portu se spolu s adresou serveru načtenou také z konfiguračního souboru metodou vrat_host(), vloží jako parametry metody getaddrinfo, která převede informace o cílovém PC, v tomto případě serveru, do dynamicky alokovaného linked listu.

```
ret_msg = getaddrinfo(obj_sett.vrat_host().c_str(), vysl, &hints,
&result);
```

Dále následuje inicializace semaforu a také nastavení časového limitu, který bude použit pro první připojení k serveru a poté bude přepsán hodnotou, kterou od něj přijme. Pro rozložení zátěže na server je použitý limit 20 sekund a k němu je přičtena náhodná hodnota od 0 do 7 sekund. Tímto způsobem by se měli najednou zapnuté počítače (funkcí wake on lan) přihlašovat k serveru v různé okamžiky.

Následuje nekonečný cyklus while, ve kterém se klient zkouší připojit na všechny adresy, které získal funkcí getaddrinfo v cyklu for.

```
for(ptr=result; ptr != NULL; ptr=ptr->ai_next)
```

V tomto cyklu se pomocí příkazu socket vytvoří IPv4, nebo IPv6 soket, dle toho jakou verzi systém používá.

```
sock_client = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);
```

Proměnnou sock_client ověříme, zda nedošlo k chybě. Dále přepneme soket do neblokovacího režimu – nejprve funkcí fnctl s makrem F_GETFL, které načte příznaky zadaného soketu do proměnné nastaveni_soc. Dalším příkazem fnctl, ve kterém se použije makro F_SETFL budou nastaveny nové příznaky – v tomto případě ty co byly získány v předchozím kroku a k nim připojený O_NONBLOCK, který nastavuje neblokovací režim soketu.

```
nastaveni_soc = fcntl(sock_client, F_GETFL, 0);
if(fcntl(sock_client, F_SETFL, nastaveni_soc | O_NONBLOCK) == -1)
```

Po provedení se kontroluje návratová hodnota funkce, která je v případě chyby -1. Pokud nastane chyba, musí být smazán chybně nastavený bit popisovače fd.

```
FD_CLR(sock_client, &sada);
```

Dále funkcí FD_ZERO vyčistíme set a následně ho funkcí FD_SET() nastavíme na potřebné parametry, které jsme získali funkcí fcntl() a poté definujeme interval, po kterém se bude klient pokoušet k serveru připojit v případě, že neběží.

```
FD_ZERO(&sada);
FD_SET(sock_client, &sada);
timeout.tv_sec = 10;
timeout.tv_usec = 0;
```

Samotné připojení je realizováno funkcí connect(), které jsou předány v předchozích krocích nastavené parametry.

```
socket_error = connect(sock_client, ptr->ai_addr, (int)ptr-
>ai_addrlen);
```

Proměnná socket_error, slouží k ověření zda, nedošlo k chybovému stavu. Pokud ano dojde opět k vyčištění špatně naplněných struktur a k uzavření soketu metodou *close()*. Pokud nenastane žádný chybový stav, pokračuje se funkcí *select()*, která vrací počet soketů, na kterých došlo k žádané události. Tedy jsou aktivní. Funkce vrací chybové stavy v záporných číslech, nebo 0 což znamená, že není aktivní žádný soket, případně došlo k chybě.

```
socket_error = select(sock_client + 1, NULL, &sada, NULL, &timeout);
```

Poté se provede funkcí getsockopt(), kde je první parametr identifikátorem soketu a SOL_SOCKET na druhém místě značí obecné vlastnosti o soketu a nakonec třetím parametrem dojde k samotnému specifikování konkrétní informace – SO_ERROR, která je nulová pokud se podařilo připojit.

```
getsockopt(sock_client, SOL_SOCKET, SO_ERROR, (char*)(&gg), &k);
```

Dále přepneme soket zpět do blokovacího režimu a tento krok ověříme na návratovou hodnotu -1, která značí neúspěch. V případě neúspěchu pokus o zablokování ještě jednou opakujeme.

```
if(fcntl(sock_client, F_SETFL, nastaveni_soc) == -1)
```

Po úspěšném navázání spojení se serverem, je nutné zvolit druh komunikace – tedy vybrat mezi komunikací zabezpečenou pomocí OpenSSL, nebo komunikací šifrovanou bitovým posunem. Tento výběr je realizován ověřením načtené hodnoty z konfiguračního souboru – stejně jako v případě inicializace OpenSSL knihoven.
Nejprve bude popsána komunikace zabezpečená pomocí OpenSSL a následně komunikace „nešifrovaná“, kde jsou předávané zprávy kódovány pomocí bitového posunu – interní šifrou.

3.7.1 OpenSSL komunikace

Nejprve dojde k volbě typu komunikace – v tomto případě, jde o klientskou aplikaci a bude tedy zvolen klientský typ komunikace ve verzi 3 a také bude načtena hloubka ověřování certifikátů z konfiguračního souboru. Pro korektní spojení musí i server podporovat konkrétní metodu SSL verze 3.

```
ctx = SSL_CTX_new(SSLv3_client_method());
SSL_CTX_set_verify(ctx, obj_sett.vrat_ssl_dept_check(), NULL);
```

Následně se dle nastavené hloubky ověřování certifikátů ověří i klientské certifikáty (certifikát a klíč) – standardně se ověřuje jen certifikát serveru. Hodnotou v proměnné ret_msg se ověřuje úspěšnost načtení certifikátu.

```
ret_msg=SSL_CTX_use_certificate_file(ctx,obj_sett.return_cert().c_st
r(), SSL_FILETYPE_PEM);

ret_msg=SSL_CTX_use_PrivateKey_file(ctx,obj_sett.return_key().c_str(
), SSL_FILETYPE_PEM);
```

A dále probíhá opět dle konfigurace kontrola tohoto načteného klíče metodou `SSL_CTX_check_private_key(ctx).` Následně se vytvoří nová struktura příkazem SSL_new(), která bude použita v této SSL relaci. Opět se ověřuje, zda se vytvoření struktury povedlo. Poté dojde ke svázání vytvořené SSL struktury s inicializovaným soketem, kde se hodnota proměnné err opět ověřuje na chybový stav.

```
err = SSL_set_fd(ssl_klient, sock_client);
```

Po tomto spojení nastavené OpenSSL struktury se soketem, bude proveden samotný SSL_connect(ssl_klient), který má jako parametr naplněnou SSL strukturu. Po úspěšném připojení se nejprve zjišťuje použitá šifra příkazem SSL_get_cipher(), a následně se získává certifikát od serveru.

```
cert = SSL_get_peer_certificate(ssl_klient);
```

Po korektním přijetí certifikátu se ještě ověří jeho platnost a pravost příkazem SSL_get_verify_result(), který by měl vrátit odpověď X509_V_OK pokud je vše v pořádku.

Tímto bylo dokončeno vytvoření zabezpečeného OpenSSL spojení a klient může komunikovat se serverem přes funkce SSL_write() a SSL_read(). Jako poslední krok se provede uvolnění struktury X509 příkazem X509_free(cert), která již není potřeba, samozřejmě do dalšího průchodu.

3.7.2 Průběh komunikační části

Komunikační část je u obou metod komunikace obdobná jen jsou použity jiné metody pro příjem a odeslání dat. Proto se budu při popisu věnovat OpenSSL komunikaci, kde budu navazovat na již popsané navázání OpenSSL spojení a poté při popisu nešifrované komunikace zmíním pouze způsob odesílání a příjmu dat, protože až na tyto metody jsou tyto části shodné.

V samotné komunikaci dojde nejprve k vyprázdnění bufferu, aby se operace neopakovali, nebo nedocházelo k jiným chybám. Poté se uloží obsah operace info() do proměnné *todo* a provede se zjištění délky této proměnné metodou length(). Metoda info() vypadá následovně:

```
int navraceno = system("who -u");
if(navraceno == 0)
{
   return "SYSTEM(Linux)";
}else{
   return "Prihlaseno(Linux)";
```

Následuje odeslání takto zjištěných dat serveru příkazem SSL_write(), kde ssl_klient reprezentuje použitou SSL strukturu, todo.c_str() jsou data odesílaná serveru převedená na datový typ char a nakonec délka těchto dat. V případě, že se návratová hodnota této operace nerovná délce odesílaných dat, tak došlo k chybě přenosu.

```
ret_msg = SSL_write(ssl_klient, todo.c_str(), todo.length());
```

Po odeslání informací dojde na samotné načtení odpovědi od serveru, která je připravena pro konkrétní pracovní stanici příkazem SSL_read(). Parametry jsou v tomto případě obdobné – SSL struktura, buffer datového typu char, do kterého bude odpověď uložena a číselná hodnota reprezentující maximální velikost odpovědi.

```
ret_msg = SSL_read(ssl_klient, buf, sizeof(buf));
```

Po přijetí odpovědi následuje její zpracování klientem. Neboli dojde k jejímu „rozparsování“. Od serveru přijdou dva druhy informací a to čas, za který se má klient opět připojit k serveru a operace, kterou má provést. Tyto informace jsou od sebe odděleny znakem „|“. A jejich zpracování probíhá následovně.

```
received_data = strtok(CommBuffer, "|");
connection_timeout = atoi(received_data);
received_data = strtok(NULL, "|");
received_operation = received_data;
```

Pro klienta nejdůležitější informace se bude po tomto zpracování příchozích dat nacházet v proměnné received_operation, se kterou se bude dále pracovat. Ověříme zda, jejím obsahem není řetězec „nic“, což by znamenalo, že klient počká po přijatý časový interval a znovu se připojí k serveru. V opačném případě se vytvoří vlákno, které provede požadovanou operaci. Vláknu se nejprve předají parametry, se kterými bude ve volané metodě pracovat, tedy název operace a semafor. Poté se vlákno vytvoří a zvedne se počítadlo vláken.

```
pthread_mutex_lock(&semafor_id);
vlakno_arg[pocet_vlaken].str_operace = operation;
vlakno_arg[pocet_vlaken].connectt = this;
vlakno_arg[pocet_vlaken].semafor_id = semafor_id;

pthread_create(&id_vlakna[pocet_vlaken], NULL, thread_function,
&vlakno_arg[pocet_vlaken]);
pthread_detach(id_vlakna[pocet_vlaken]);
pocet_vlaken++;
pthread_mutex_unlock(&semafor_id);
```

Po provedení této operace či v případě přijetí řetězce „nic“ se pokračuje úklidem použitých struktur a uzavřením soketu. Následně klient čeká po přijatý čas na opětovné připojení k serveru.

```
SSL_shutdown(ssl_klient);
SSL_free(ssl_klient);
SSL_CTX_free(ctx);
shutdown(sock_client, SHUT_RDWR);
close(sock_client);
sleep(received_timeout);
```

3.8 Nešifrovaná komunikace

Nešifrovaná komunikace je realizována funkcí string clear_communication(string data). Tato metoda má na starosti odeslání dat serveru – ty dostane jako parametr a zároveň data od serveru přijme a vrátí je po svém skončení. Na začátku této metody jsou definovány proměnné typu char pro šifru a data, která budou přijata od serveru.

Prvním krokem je přijetí šifry od serveru, kterou načte, do proměnné typu char. Po přijetí dat se kontroluje, zda proběhl přenos dat v pořádku.

```
read(sock_client, prijato, sizeof(prijato));
```

V případě chyby přenosu se provádí ve všech případech zápis do logu a uzavření soketu. Metoda v takovém případě vrací řetězec „Error“

```
error("Cannot read data from server!");
shutdown(sock_client, SHUT_RDWR);
close(sock_client);
return "Error";
```

Následuje uložení přijatých dat do proměnné cipher. Následně se provede zakódování odesílaných dat, dle přijaté šifry, pomocí metody codding(). Praktiky se bere znak po znaku v cyklu a šifruje se.

```
for(int i = 0; i < todo.length(); i++)
{
   todo[i] = codding(todo[i], cipher[j]);
   if(len == j)
   {
          j = 0;
   }else{
          j++;
        }
}
```

Po zašifrování jsou data připravena k odeslání serveru. Odeslání je realizováno metodou write(), která dostane parametry jako soket, odesílaná data a číselnou proměnnou reprezentující délku odesílaných dat.

```
write(sock_client, proved.c_str(), proved.length());
```

Po odeslání dat serveru následuje vynulování používaných proměnných a následné příjem dat, které tentokrát obsahují standardně přijímaný řetězec času a názvu operace. Než se mohou přijatá data vrátit zpět do metody connect_to_server() je nutné provést jejich dekódování do čitelné podoby. Toto dekódování se provádí ve stejném cyklu jako kódování jen je místo metody codding() použita metoda decodding(). Po dekódování následuje ještě uzavření použitého soketu a navrácení dešifrovaných dat.

Dalším postupem je zpracování těchto dat v metodě connect_to_server() a jejich vyhodnocení, které je shodné s již popsaným postupem v případě openSSL komunikace.

3.8.1 Třída Operations.cpp

Tato třída je volána ve vláknu z metody *connect_to_server()* ve chvíli, kdy od serveru přijdou data s názvem určité operace k provedení. Zde se roztřídí a provede se zavolání odpovídající metody ze třídy execution_methods.cpp, exec představuje objekt dané třídy.

```
if(oper == "vypnout")
{
   exec.shutdown();
}
else if(oper == "restartovat")
{...}
```

3.8.2 Třída Execution_methods.cpp

V této třídě je umístěn samotný výkonný kód prováděný v případě určité operace. Již popsaná je operace info v oddílu popisu metody connect_to_server(). Dále třída obsahuje metody shutdown(), shutdown_force(), reboot() a reboot_force(), které dělají přesně to co je z jejich názvu patrné.

Metoda *shutdown ()*, provede vypsání upozornění o plánovaném vypnutí na všechny aktivní konzole, což zajistí skript „wall“. Dále se provede ověření, zda je uživatel přihlášen zavoláním metody info() a ověřením její návratové hodnoty. Pokud není nikdo přihlášen, bude provedeno okamžité vypnutí příkazem system(„halt“); a v opačném případě se zavolá grafická nadstavba s parametrem -s a nabídne uživateli možnost odložení.

```
system("echo 'System bude za 3 minuty na prikaz spravce vypnut...\n
Pro odlozeni provedte: touch /tmp/bcontrolc' | wall");
if(info() == "SYSTEM(Linux)")
{
   system("halt");
}else{
   system("java -jar /opt/bcontrolc/bin/BconGUI.jar -s");
}
```

Obdobně je to v případě metody reboot(), rozdíl je pouze u metod reboot_force() a shutdown_force(), kde jsou přímo volány system(„reboot“) a system(„halt“).

3.8.3 Třída configuration.cpp

Třída obsahuje sekci, která bude načtena z konfiguračního souboru a definice načítaných proměnných. Tato třída obsahuje dvě funkce – cb_validate_bookmark() a load_config().

cb_validate_bookmark()

Funkce kontroluje sekci, která byla vložena jako parametr. Kontroluje se, zda není daná sekce prázdná. V případě chyby vrací -1.

load_config()

Funkce obsahuje strukturu s proměnnými, které mají být načteny. V této ukázce je zobrazena samotná definice struktury a jedné proměnné s názvem server. Obsaženy jsou ještě definice datových typů a implicitní hodnota, na kterou bude proměnná nastavena, dokud nebude přepsána hodnotou z konfiguračního souboru.

```
static cfg_opt_t bookmark_opts[] = { CFG_STR((char *)"server", (char
*)"localhost", CFGF_NONE),
….};
```

Po této základní definici proměnných, které budou načítány, se provede inicializace a následně validace zadané sekce metodou *cfg_set_validate_func()*, která má jako parametr inicializovanou sekci. Následuje ověření integrity konfiguračního souboru funkcí *cfg_parse()*, kde ověřujeme návratovou hodnotu funkce na chybové stavy.

Po ověření integrity souboru započne samotné plnění proměnných hodnotami z konfiguračního souboru.

```
host = cfg_getstr(section_part, "server");
```

Po naplnění proměnných dojde k uvolnění inicializované struktury cfg, protože již není dále potřeba. Nyní proběhne ověření, zda cesta k certifikátům, která je uvedena v konfiguračním souboru, je správná.

```
FILE *f = fopen(lokace.c_str(), "r");
```

Pokud proměnná je proměnná f po tomto příkazu NULL je vypsána chyba o nenalezení certifikátu v opačném případě se pokračuje ověřováním ostatních certifikátů.

Následně jsou ověřeny hodnoty proměnných *ssl_dept_check* a *security*, které reprezentují hloubku ověřování certifikátů a úroveň zabezpečení, zda nejsou nastaveny na nesmyslné hodnoty.

3.8.4 Třída configuration_methods.cpp

Tato třída obsahuje definice funkcí, které jsou volány třídou communication.cpp. Názvy těchto proměnných se musejí shodovat s odpovídajícími proměnnými ve třídě configuration.cpp.

```
string Configuration::return_host()
{
   return host;
}
```

3.8.5 Požadavky pro kompilaci

Program byl kompilován pomocí souboru makefile na systému Ubuntu 10.04, kde byly doinstalovány knihovny liblog4cpp5, libssl-dev, libconfuse-dev.

Tímto byly popsány všechny třídy, které se podílí na běhu programu a další část bude věnována grafické aplikaci pro systém Linux.

4. Grafická aplikace pro systém Linux

Jak již bylo zmíněno v části (3.1) věnované volbě programovacího jazyka, byl pro tvorbu grafické nadstavby aplikace použit programovací jazyk JAVA s grafickou knihovnou Swing.

Jedná se o relativně jednoduchou aplikaci, která obsahuje pouze dvě třídy. Jedna slouží pro samotné spuštění a z druhé třídy jsou volány metody, které budou provedeny, na základě spouštěcího parametru.

4.1 Rozpis parametrů programu

Program v linuxové verzi je použit pouze s dvěma různými parametry

- r program zobrazí upozornění na nadcházející restartování počítače, s možností odložení
- s program zobrazí upozornění na nadcházející vypnutí počítače, s možností odložení

Do programu je možné samozřejmě snadno implementovat další funkce jako, hibernaci a uspání cílového počítače, ale pro použití s tímto systémem nebyla tato funkčnost vyžadována. Hlavně s ohledem na to, že uspávání počítače stále není nejspolehlivější operace a uživatel by mohl přijít o rozdělanou práci.

Spouštění tohoto programu se provede příkazem „java /cesta k programu/BconGUI.jar -parametr“, kde se použije jeden s výše uvedených parametrů. Při spuštění programu bez zadání parametru nebude provedeno nic.

4.2 Popis kódu

V této části se budu věnovat popisu vypnutí počítače s možností odložení, realizovaného přes výše popsanou JAVA aplikaci.

Jak již bylo napsáno v rozpisu parametrů, pro vypnutí počítače je nutné spustit tento program s parametrem „-s“, který se vyhodnotí ve spouštěcí třídě Run.java. Vyhodnocení probíhá v podmínce if.

Po zjištění parametru se spustí příslušná metoda, ze třídy Methods.java, v tomto případě to bude metoda Shutdown (). Tato metoda nejprve vytvoří nové vlákno, které bylo vytvořeno implementací rozhraní runnable. Tedy v tomto konkrétním případu vypnutí, bude vytvořeno vlákno a jako parametr mu bude předáno rozhraní RS.

```
Thread t = new Thread(RS);
```

Samotná metoda Shutdown(), dále pokračuje vytvořením dialogového okna, kterému jsou zadány parametry JOptionPane.YES_NO_OPTION. Tyto parametry zaručí, že bude vykreslené okno obsahovat dvě tlačítka ANO a NE, jazyk tlačítek bude dle jazykové verze systému.

```
int n = JOptionPane.showConfirmDialog(…);
```

Poté se ověří návratová hodnota proměnné „n“, pokud uživatel zvolí tlačítko ANO, program počká do uběhnutí tříminutového intervalu vlákna a počítač bude vláknem vypnut.

V případě volby NE bude zrušeno vlákno implementující rozhraní RS a dojde k uspání hlavního vlákna programu po dobu třiceti minut. Po uplynutí této doby, bude opět spuštěna metoda Shutdown() a celí proces se bude opakovat.

Pro případ, že uživatel nemá spuštěné grafické prostředí je implementováno ověření vytvoření souboru „bcontrolc“ v adresáři tmp. Tímto způsobem bude vypnutí, nebo restartování počítače odloženo a soubor bude smazán. V situaci, kdy soubor nebude přítomen se systém standardně vypne, nebo restartuje.

Pokud uživatel nezvolí žádnou možnost v dialogovém okně, tak po uplynutí 3 minutového intervalu se testuje již zmíněná přítomnost souboru bcontrolc adresáři tmp, pokud je soubor přítomen, uspí se vlákno na 30 minut a opět se zavolá metoda Shutdown(). V situaci, že soubor „bcontrolc“ v adresáři tmp neexistuje, provede se okamžité vypnutí počítače.

Naprosto obdobná situace nastane v případu se zadaným parametrem –r, jen bude počítač místo vypnutí restartován.

Tímto by byl dokončen popis části systému věnovaného systému Linux. Nyní se budu věnovat popisu rozdílů mezi Linux a Windows verzí klientské aplikace a grafické nadstavby.

5. Tvorba klientské aplikace pro systém Windows

Při tvorbě Windows verze klientské aplikace, byly z velké části použity zdrojové kódy Linux verze aplikace. Nejprve byl založen projekt v programu Visual Studio 2008, který využíval Visual C++, konkrétně „Windows service“.

5.1 Struktura programu

Z důvodu, že Windows verze klienta vychází z velké části z Linux verze, tak jsou obdobné i struktury tříd. Proto zde popíši jen ty, kde jsou nějaké změny. Ve Windows verzi zastává funkci hlavičkového souboru soubor stdafx.h a je prakticky shodný se souborem functions.h, jen byly nahrazeny knihovny typické pro systém Linux, alternativami systému Windows.

AssemblyInfo.cpp

Obsahuje informace o nastavení a parametrech toho sestavení.

BcontrolcWinService.cpp

Třída plní funkci třídy main.cpp v systému Linux, ale zde se s tohoto třídy spouští třída _tmain()

- void hlavni_metoda()
- int _tmain()

Communication.cpp

Třída obsahuje shodné metody s Linux verzí.

Config.cpp

Třída Configuration.cpp musela být v systému Windows přejmenována na Config.cpp z důvodu shody názvu se systémovou třídou. Jinak je shodná s Linux verzí.

ConfigurationMethods.cpp

Třída obsahuje shodné metody s Linux verzí.

ExecutionMethods.cpp

Tato třída obsahuje veškeré metody ze systému Linux a ještě jsou v ní doplněny tyto:

- BOOLEAN DisplayLocalLogons()
- Void get_permission()
- Void return_permission()

- BOOL do_task()

Operations.cpp

Tato třída obsahuje metody shodné s Linux verzí.

5.2 Přidání OpenSSL knihoven

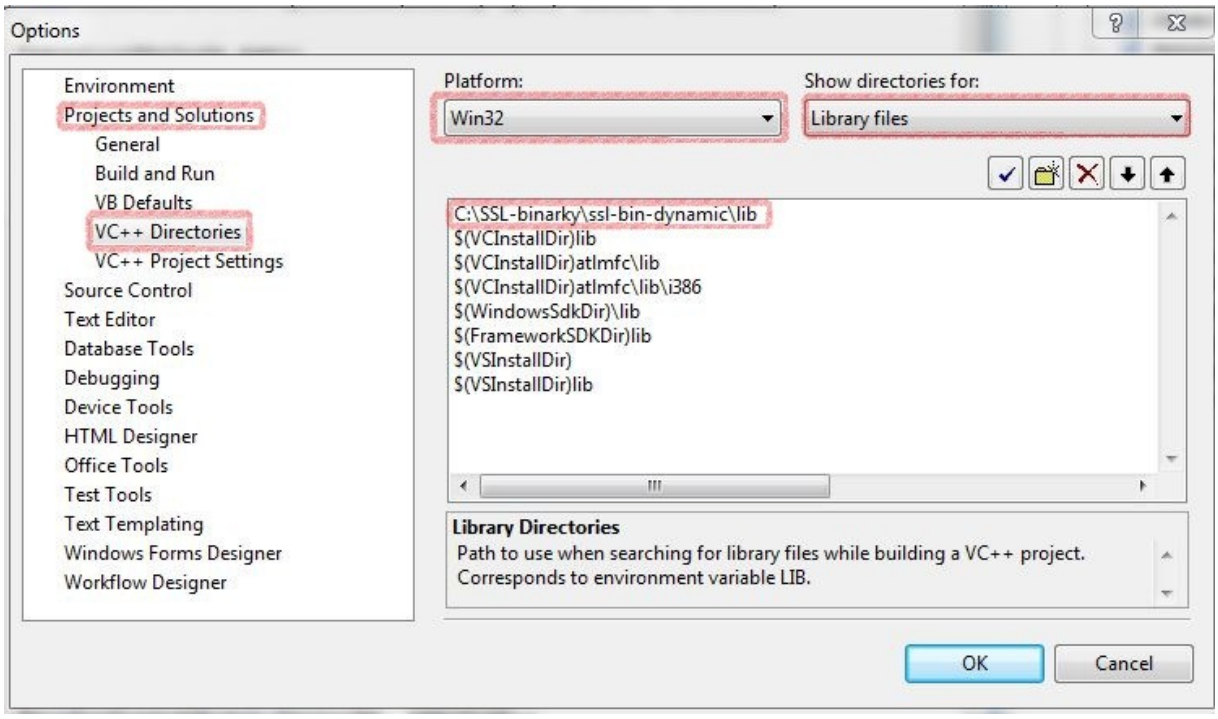
Jelikož jsou použity v době kompilace nejnovější openSSL knihovny, které byly zkompilovány přímo ze zdrojových kódů, je potřeba ve Visual Studiu 2008 nastavit k těmto knihovnám cestu ve VC++ Directories.

Tools -> Options

V dialogovém okně nastavení je nutné se přesunout do:

Project and Solutions -> VC ++ Directories

Následně budou přidány cesty k „Library files“ a „Include files“ pro platformu Win32.



Obr. č. 4 - Dialog VC++ Directories

Obdobně to bylo provedeno v případě Include files.

5.3 Vysvětlení kódu

Hlavní struktura služby pro systém Windows, byla vygenerována programem Visual Studio 2008. Základní parametry služby byly nastaveny ve vygenerovaném souboru BcontrolWinService.h, kde základní konstruktor volá metodu InitializeComponent(), ve které je zakázáno pozastavení služby.

Dále jsou vygenerovány metody OnStrart() a OnStop() do metody OnStart(), bylo doplněno ověření, zda neběží vlákno aplikace a poté byla vláknu předána hlavní metoda a následně bylo spuštěno.

```
virtual void OnStart(array<String^>^ args) override
{
   if ( (workerThread == nullptr) || ((workerThread->ThreadState &
   (System::Threading::ThreadState::Unstarted |
   System::Threading::ThreadState::Stopped)) !=
   (System::Threading::ThreadState)0) )
   {
   workerThread = gcnew Thread( gcnew ThreadStart(
   this,&NetconcWinService::hlavni_metoda ) );
   workerThread->Start();
   }
}
```

V metodě OnStop(), se provede pouze zrušení vlákna aplikace. Tuto operaci není nutné nijak ošetřovat, protože systém Windows volbu Stop vůbec neumožní, pokud služba neběží.

```
workerThread->Abort();
```

Samotná metoda InitializeComponent(), obsahuje nastavení parametrů služby. Parametr CanStop – umožňuje zastavení služby, AutoLog – povoluje automatické ukládání událostí do logu systému Windows, ServiceName – určuje jméno služby ve správci služeb systému Windows.

```
void InitializeComponent(void)
{
   this->components = gcnew System::ComponentModel::Container();
   this->CanStop = true;
   this->CanPauseAndContinue = false;
   this->AutoLog = true;
   this->ServiceName = L"Bcontrolc";
}
```

Dalším vygenerovaným souborem je soubor AssemblyInfo.cpp, ve kterém jsou obsaženy informace o Verzi assembly, datu vytvoření a také velmi důležitá věc a to povolení tzv. „Unmanaged kódu“ tedy nativního C++ kódu.

```
[assembly:SecurityPermission(SecurityAction::RequestMinimum,
UnmanagedCode = true)];
```

Díky tomuto je možné použití velké části zdrojových kódů, které byly napsány pro systém Linux.

Poslední vygenerovanou částí je BcontrolcWinService.cpp, který je obdobou klasického C++ main.cpp. Obsahuje tedy metodu „Hlavni_metoda“ která je přiřazena hlavnímu vláknu služby jako parametr v již popsaném souboru BcontrolcWinService.h.

```
void NetconcWinService::hlavni_metoda()
{
   Config *obj_sett = new Config();
   obj_sett->load_config();
   Communication *obj_spojeni = new Communication();
   obj_spojeni->connect_to_server(*obj_sett);
}
```

V této metodě se vytvoří objekt, přes který se bude přistupovat k jednotlivým proměnným konfiguračního souboru a spustí se metoda connect_to_server(), která je detailně popsána v části věnované klientské aplikaci pro systém Linux.

Dále je v této třídě zpracování spouštěcích parametrů a postup instalace aplikace parametrem „-install“, které byly vygenerovány při vytvoření služby.

Dále byly do souboru stdafx.h nakopírovány používané knihovny a případné nepodporované nahrazeny ekvivalentními Windows knihovnami. A také byl doplněn soubor classes.h, který má shodnou funkci jako u alternativního souboru pro systém Linux.

5.4 Změny v kódu

Nejprve bude vysvětlena třída communication.cpp, kde jsou hlavní rozdíly v soketové komunikaci a to z důvodu rozdílných volání v systému Linux a systému Windows.

5.4.1 Změny v metodě clear_communication

Odesílání dat v systému Linux bylo provedeno příkazem:

```
write(sock_client, todo_data.c_str(), proved.length());
```

v systému windows je alternativou:

```
send(socket_client, todo_data.c_str(), todo_data.length(),0);
```

Tato funkce má navíc parametr int flags, kterým lze specifikovat, jak bude odeslání provedeno – standardně se používá 0. Obdobně je to i u obdržení dat, kde se volání read() systému Linux, nahradilo

```
recv(socket_client, prijato, sizeof(prijato),0);
```

5.4.2 Změny v metodě connect_to_server

V metodě connect_to_server pro systém Windows, je nejprve nutné inicializovat struktury pro vytvoření soketu. Dále je nutné vybrat verzi soketů, která bude pro komunikaci použita, v tomto případě byla zvolena poslední verze 2.2

```
wVersionRequested = MAKEWORD(2, 2);
```

Následně byly parametry předány funkci WSAStartup(), která provedla nahrání knihovny Winsock.dll pro použití v programu. Ověřuje se, zda nahrání proběhlo v pořádku – pokud proběhne bez problémů, vrací hodnotu 0.

Po inicializaci následuje ověření, zda je daná verze v systému použitelná

```
if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
```

Pokud není, dojde k odstranění struktur a vyčištění paměti příkazem WSACleanup().

Následuje rozdílná inicializace semaforu, kde se v systému Windows jako parametry uvádí bezpečnostní atributy a název tohoto semaforu.

```
CreateMutex(NULL, true, L"Citac");
```

Při vytváření soketu v cyklu while, jsou rozdílné návratové hodnoty – v případě systému Linux jsou navráceny číselné hodnoty, v systému Windows se ověřují přímo stavy.

```
if(socket_client == INVALID_SOCKET)
```

Přepnutí soketu na neblokovací režim je v systému Windows provedeno příkazem, kde parametr FIONBIO určuje neblokovací typ soketu. Toto nastavení bylo v systému Linux provedeno příkazem fnctl s příznakem O_NONBLOCK.

```
if(ioctlsocket(socket_client, FIONBIO, &iMode) == SOCKET_ERROR)
```

Připojení probíhá v obou systémech shodně příkazem connect(), jen se liší návratové (chybové stavy) hodnoty. Shodná situace i u funkce select().

V případě OpenSSL komunikace jsou využívány standardní OpenSSL volání, jak pro jejich inicializaci, tak pro samotné odesílání a příjem dat, tedy tyto části jsou v obou systémech shodné.

Rozdílná je část tvorby vlákna, kde je využita konvence pro systém Windows. Následně se liší ještě uzavření soketu místo volání shutdown() v systému Linux, je použito volání closesocket().

Nyní budou popsány metody, které nejsou obsaženy v klientské aplikaci pro systém Linux.

5.4.3 Třída ExecutionMethods.cpp

Třída ExecutionMethods.cpp je ve verzi pro systém Windows složitější z toho důvodu, že bylo nutné implementovat metody pro zvýšení práv této službě. Služba běží po většinu času s uživatelskými právy, což je praktické z pohledu bezpečnosti, ale neumožnilo by nám to vypnout ani restartovat počítač. Proto byly napsány metody pro získání a opětovné navrácení administrátorských práv službě.

5.4.4 Metoda get_permission

Pro samotné zvýšení práv slouží ve třídě ExecutionMethods.cpp metoda get_permission(). Nejprve je pomocí funkce OpenProcessToken, získán token konkrétního procesu

```
OpenProcessToken(GetCurrentProcess(),TOKEN_ADJUST_PRIVILEGES,
&token)
```

Následně je třeba získat ještě tak zvané LUID – jedná se o 64-bitový identifikátor, který je vygenerován při startu systému a je platný do restartu. Jeho zjištění bylo provedeno

```
!LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME,
 &privilegia.Privileges[0].Luid));
```

Dále bylo provedeno samotné zvýšení oprávnění funkcí AdjustTokenPrivileges()

```
AdjustTokenPrivileges(token, FALSE, &privilegia, 0, NULL, NULL);
```

Ověřením posledního chybového stavu, je zjištěno, zda proběhla operace v pořádku.

Po provedení operace vyžadující Administrátorská práva, je nutné tato práva zase navrátit. Na tuto operaci slouží metoda retur_permission()

5.4.5 Metoda return_permission

Tato metoda nastaví Privileges[0].Attributes na 0, která reprezentuje navrácení administrátorských práv.

```
privilegia.Privileges[0].Attributes = 0;
AdjustTokenPrivileges(token, FALSE, &privilegia,
0,(PTOKEN_PRIVILEGES) NULL, 0);
```

5.4.6 Metoda do_task

Tato metoda slouží pro provedení určité operace formou volání externí aplikace. Tímto způsobem bylo možné provést uživatelské odložení vypnutí, nebo restartu systému. Proměnná operace, která je parametrem této metody, slouží jako spouštěcí parametr volaného programu.

V této metodě dojde k nahrání používaných knihoven (kernel32.dll a Userenv.dll) funkcí LoadLibrary.

```
HMODULE  hmod = LoadLibrary(L"kernel32.dll");
```

Následně je třeba zjistit ID aktuálně aktivní relace Windows konzole.

```
DWORD dwSessionId = WTSGetActiveConsoleSessionId();
```

A také příznak pro tuto konkrétní relaci systému

```
WTSQueryUserToken(dwSessionId, &hToken);
```

Dalším krokem je vytvořeno prostředí, pomocí příkazu CreateEnvironmentBlock, které bude přecházet do procesu.

```
if(CreateEnvironmentBlock(&pEnv, hToken, FALSE))
{
  dwCreationFlag |= CREATE_UNICODE_ENVIRONMENT;
}
```

Z registrů je dále načtena cesta, ze které byla spuštěna služba, na tuto operaci je potřeba zvýšit oprávnění proto je zavolána metoda get_permission() a následně po provedení oprerace, jsou práva navrácena zpět. Výsledek operace je uložen do proměnné buf.

```
RegQueryValueExA(hKey,"ImagePath",0, &dwType, (BYTE*)buf,
&dwBufSize) == ERROR_SUCCESS)
```

Aby se nestalo, že se grafická nadstavba programu nespustí, z důvodu instalace programu do jiného než defaultního adresáře, je cesta ke grafické aplikaci zjišťována přímo ze služby. Zjištění cesty bylo provedeno metodou _slitpath, a následně došlo ke zpracování získaného řetězce a na konec byl přidán název grafické nadstavby pro systém Windows.

```
_splitpath( buf, node, dir, fname, ext );
```

Konečnou fází této metody je provedení příkazu CreatreProcessAsUser, která spustí danou aplikaci s přijatými parametry.

```
if ( !CreateProcessAsUser(
hToken,
clear_path,
operation,
NULL,
NULL,
```

```
FALSE,
dwCreationFlag,
pEnv,
NULL,
&si,
&pi
) )
```

Je velmi dobře vidět, že spuštění programu ze služby je v systému Windows značně náročnější než v případě systému Linux. Tento postup je univerzálně funkční na operačním systému Windows XP až Windows 7(32-bit i 64-bit edice). Základní postup byl převzat ze stránek MSDN, ale bylo nutné ho upravit, aby fungoval i pod Windows 7, protože v základu služba při spuštění aplikace havarovala.

5.4.7 Metoda DisplayLocalLogons

Tato metoda slouží k ověření, zda se cílová stanice nachází na přihlašovací obrazovce, nebo je přihlášen uživatel. Tato metoda se volá v metodě info(). Kód této metody byl z velké části převzat ze stránek MSDN.

V kódu byly provedeny drobné úpravy a zároveň byl upraven pro zjištění uživatelského jména přihlášeného uživatele.

```
if(wcscmp(domainName, L"NT AUTHORITY") != 0){
tchar_string = userName;
```

Jelikož je proměnná userName datového typu TCHAR, bylo ji nutné před odesláním serveru převést na datový typ const char. Tento převod byl proveden v metodě info(), kde se data připravují na odesílání serveru. Pro převod byla použita metoda WideCharToMultiByte, kde se převáděná proměnná tchar_username, uloží do proměnné `AnsiBuffer`.

```
WideCharToMultiByte(CP_ACP, 0, tchar_username,
wcslen(tchar_username)+1, AnsiBuffer , sizeof(AnsiBuffer), NULL,
NULL);
```

Takto převedené jméno uživatele bylo následně přidáno do výsledného řetězce „Prihlaseno Win(USERNAME)“ pomocí funkcí strcpy a strcat.

V této kapitole byly popsány změny v klientské službě pro systém Windows a následující kapitola bude věnována grafické nadstavbě pro systém Windows.

6. Grafická nadstavba systému Windows

Jako programovací jazyk pro grafickou nadstavbu systému Windows byl zvolen jazyk C#. Volba to byla poměrně snadná, protože .NET knihovny jsou standartní součástí systému Windows Vista a výše a nabízí velmi dobrý výkon pod tímto systémem. Zároveň instalační verze použitých .NET 2.0 knihoven je relativně malá a tedy nezvětšuje zbytečně instalační soubor. Do instalačního souboru jsou knihovny přiloženy s volitelnou možností instalace pro Windows XP, který nemá standardně knihovny nainstalovány.

6.1 Popis programu

Program pro interakci s uživatelem na systému Windows, se skládá obdobně jako verze pro systém Linux ze dvou tříd, kde první Program.cs, slouží k samotnému spuštění určité metody, ze třídy Operations.cs, dle spouštěcího parametru.

```
if (args.Length == 1 || args.Length == 2)
{
      cmd = args[0].Replace('/', '-');
}
```

Tímto způsobem bude nahrazeno lomítko za pomlčku u spouštěcího parametru. Následně je v bloku switch, dle spouštěcího parametru zvolena odpovídající metoda.

6.1.1 Třída Operations.cs

V této třídě jsou implementovány samotné metody, které zajistí vypnutí, nebo restartování počítače po uběhnutí časového limitu. V následujícím textu bude popsána metoda shutdown_time, která slouží k vypnutí počítače s možností uživatelského odložení.

Při spuštění metody Shutdown_time(), je jí jako parametr předána číselná hodnota reprezentující počet minut, které má čekat před vypnutím a také bool hodnota, která reprezentuje vynucení ukončení běžících úloh.

Metoda nejprve spustí vlákno, které zajistí, že pokud uživatel neprovede do tří minut žádnou akci, dojde k vypnutí počítače. Následně se vykreslí MessageBox, který dostane jako parametry zobrazovaný text, typ MessageBoxButtons.YesNo a MessageBoxDefaultButton.Button1 – čímž bude aktivní první tlačítko v MessageBoxu.

Pokud uživatel klikne na tlačítko „NE“ tak dojde k ukončení vedlejšího vlákna, které mělo hlídat tří minutový interval neaktivity a hlavní vlákno programu bude uspáno po dobu třiceti

minut. Po uplynutí této doby bude opět zavolána metoda Shutdown_time() a celý proces začne znovu.

V případě, že uživatel neprovede, žádnou akci, nebo vybere tlačítko ANO – tak program vyčká na doběhnutí tří minutového limitu a počítač se vypne.

Samotné vypnutí je provedeno příkazem:

```
UnsafeNativeMethods.Shutdown(UnsafeNativeMethods.ShutdownType.EWX_SH
UTDOWN | UnsafeNativeMethods.ShutdownType.EWX_POWEROFF,
shutdownReason);
```

kde je volána statická metoda Shutdown, která provede operaci, dle přijatého parametru.

```
internal static void Shutdown(ShutdownType type, uint reason)
```

V této metodě je provedeno získání identifikátorů běžící session Windows a následně zvýšení oprávnění pro operaci vypnutí, nebo restart počítače.

Obdobně je to u provedení operace restartování, samotné vypnutí, nebo restartování bez dotazu uživatele, se provede pouze zavoláním metody Shutdown s odpovídajícím parametrem. Do takto připraveného programu, není problém přidat další metody, které by počítač uspaly, hibernovaly nebo jen zamknuly. Stačilo by jen volat metodu Shutdown s jinými parametry, které jsou vidět v následující struktuře.

```
internal enum ShutdownType : uint
{
   EWX_LOGOFF = 0x00,
   EWX_SHUTDOWN = 0x01,
   EWX_REBOOT = 0x02,
   EWX_POWEROFF = 0x08,
   EWX_FORCE = 0x04,
   EWX_FORCEIFHUNG = 0x10,
   EWX_RESTARTAPPS = 0x40
}
```

7. Tvorba instalátorů

V této kapitole bude popsána tvorba instalačních balíčků, pro systém Microsoft Windows a Linux. Nejprve budou popsány použité technologie, kterými byly balíky vytvořeny, a následně popsán postup samotné tvorby.

7.1 Instalátor pro systém Windows

Systém Windows má standardní instalační balíky ve formě spustitelného .exe souboru. Proto bylo bráno v potaz, aby výstupem této práce byl jeden .exe soubor, který provede nakopírování veškerých souborů a certifikátů do zvolených umístění, nainstaluje potřebné knihovny, službu přidá do systému a spustí.

Pro vytvoření samotného instalátoru byl zvolen nástroj NSIS (Nullsoft Scriptable Install System). Jedná se o OpenSource skriptovací program pro tvorbu instalačních balíčků systému Windows. Má velmi široké možnosti nastavení a jdou jím provést prakticky libovolné úkony související s instalací. Není proto výjimečné, že je tento program využit pro tvorbu instalátorů rozsáhlých profesionálních programů. Firma Nullsoft, která je autorem tohoto nástroje, jej používá i pro tvorbu instalátoru svého multimediálního přehrávače WinAmp. Možnosti nástroje NSIS jdou ještě dále rozšířit pomocí zásuvných modulů.

Pro tvorbu instalátoru je potřeba napsat příslušný skript. Ve skriptu budou specifikovány veškeré soubory, které se při instalaci nakopírují na cílový počítač, dále jsou zde také operace, které budou provedeny, například instalace podpůrných knihoven. Veškeré operace budou popsány v části věnované popisu instalačního skriptu.

7.1.1 Instalační skript

Základem bude skript Bcontrolc skript.nsi, ve kterém bude celá struktura instalace popsána. Jako téma vzhledu instalátoru, byl zvolen skin MUI2.nsh, který nabízí velmi přehledné a jednoduché rozhraní.

Dále byl specifikován název výstupního soubodu v proměnné OutFile a také přednastavena výchozí cesta do které bude program nainstalován, proměnnou InstallDir.

Velmi důležitou položkou je specifikace požadavku na administrátorská práva v systému. Tímto bude v operačních systémech Microsoft Windows Vista a výše se zapnutým UAC(uživatelské řízení účtů) zobrazen požadavek na zvýšení uživatelských práv a instalace

bude pokračovat. Bez této úpravy by instalace na těchto systémech selhala s chybovým hlášením.

```
RequestExecutionLevel Admin
```

Další sekcí v souboru jsou stránky, které budou v instalátoru zobrazeny. Takto lze přidat dle potřeby libovolný počet kroků v instalátoru. Dokumentaci, k tomuto „Modern User Interface“ lze najít na stránkách.

Následují samotné sekce instalátoru, kde hlavní je sekce „BcontrolcBase“, ve které budou nakopírovány hlavní části programu na zvolené umístění. Tato sekce je tak zvaně trvalá – to znamená, že nejde odznačit. Definice byla provedena přidáním řádku SectionIn RO.

```
Section "BcontrolcBase"
SectionIn RO
SetOutPath "$INSTDIR"
```

Další jsou tentokrát již volitelné sekce, které reprezentují instalaci OpenSSL a .NET Framework 2.0 knihoven. Při jejich instalaci bylo dbáno na co nejnižší „obtěžování“ uživatele jsou tedy instalovány s volbou quite.

```
ExecWait '$INSTDIR\dotnetfx.exe /q:a /c:"install /q"'
```

Poslední částí v instalačním oddílu je sekce „Finish“, která provede instalaci podpůrných knihoven, pokud byly zvoleny a také spustí .bat soubor, který provede registraci služby a její spuštění.

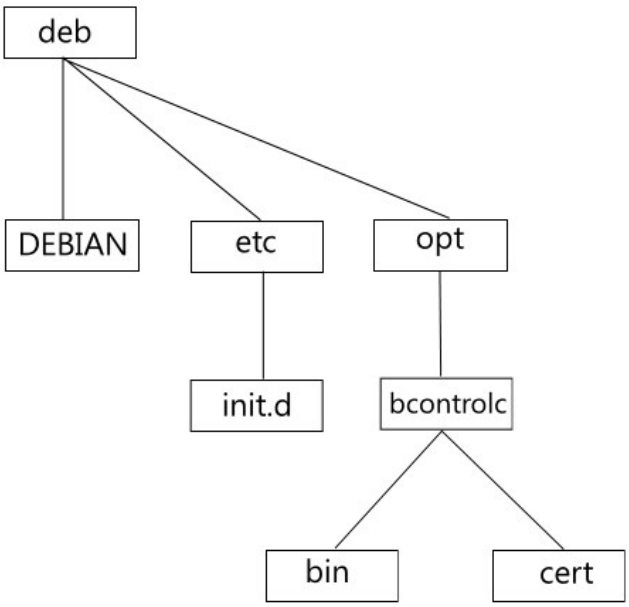
Důležitou částí instalačního skriptu je část, která vytvoří odinstalační spustitelný soubor. V této části je spuštění .bat souboru, který zastaví službu a následně ji odebere ze správce služeb. Po této operaci následuje smazání veškerých souborů, které byly při instalaci nakopírovány do počítače. Také dojde k odstranění záznamu o programu z registrů počítače.

Samotná kompilace skriptu probíhá buď vyvoláním kontextové nabídky a volbou Compile NSIS skript, nebo přetažením skriptu do okna NSIS nástroje. Veškeré soubory, které mají být přítomny v instalátoru, musí být ve shodné složce jako kompilovaný sktipt.

7.2 Instalátor pro systém Linux

Pro operační systémy Linux, postavené na bázi distribuce Debian což je i distribuce Ubuntu, pro kterou byl primárně systém vyvíjen, je standardem instalační .deb balíček.

Nejdříve bylo nutné vytvořit adresářovou strukturu, do které se následně umístily požadované soubory.



obr. č. 5 Adresářová struktura

7.2.1 Obsah složek

Zde bude vypsán obsah složek včetně účelu daných souborů.

Složka DEBIAN

- Conffiles – obsahuje cestu ke konfiguračnímu souboru a při updatu aplikace nabídne jeho ponechání.

- Control – v tomto souboru jsou definovány závislosti na knihovnách, verze aplikace, požadovaná architektura a informace o autorovi.

- Postinst – jedná se o skript, který se provede při instalaci aplikace. Provede přidání daemona, do procesů startujících po startu počítače a poté ho i spustí.
- Postrm – tento skript má na starosti odebrání záznamu o aplikaci ze souboru update.rc.d
- Prerm – tento skript provede zastavení běhu daemona, aby mohl být odebrán
- Md5sums – obsahuje kontrolní MD5 součet, který je vytvořený před tvorbou .deb balíčku níže uvedeným příkazem.

Složka etc

- Bcontrolc.conf – tento soubor obsahuje konfiguraci programu

Složka bcontrolc

- Bcontrolc – binární soubor aplikace

Složka bin

- BconGUI.jar – binární soubor grafické nadstavby v jazyce JAVA

Složka cert

- Authority.pem
- Client.key
- Client.pem

Složka init.d

- Bcontrolc – spouštěcí skript aplikace, obsahuje definici startu i zastavení aplikace

Složky, které nejsou zahrnuty v tomto přehledu neobsahují žádné soubory jen další podsložky, které jsou vyobrazeny na obr. č.5.

Po připravení výše popsané adresářové struktury bylo provedeno samotné vytvoření .deb balíku příkazy:

- `find * -type f ! -regex '^DEBIAN/.*' -exec md5sum {} \; > DEBIAN/md5sums`

tento příkaz vytvořil MD5 kontrolní součet

- `dpkg-deb -b deb bcontrolc_verze_i386.deb`

tento příkaz vytvoří samotný deb balíček zadaného názvu.

7.3 Konfigurace programu

Po instalaci je nutné upravit konfigurační soubor bcontrolc.conf, který se nachází:

Windows: C:\Program files\Bcontrolc\bcontrolc.conf
Linux: \\etc\bcontrolc.conf

```
server = "IP adresa serveru"
port = 8888
ssl = true
```

Případně, pokud bude program provozován v nešifrovaném režimu.

```
server = "IP adresa serveru"
port = 8889
ssl = false
```

Další sekce konfiguračního souboru není nutné standardně měnit, pouze pokud budou přesunuty certifikáty do jiného umístění, musí k nim být nastaveny odpovídající cesty.

Samozřejmě musí být shodné nastavení i v konfiguračním souboru serveru, jinak by nebylo možné vytvořit spojení.

8. Webová ovládací aplikace

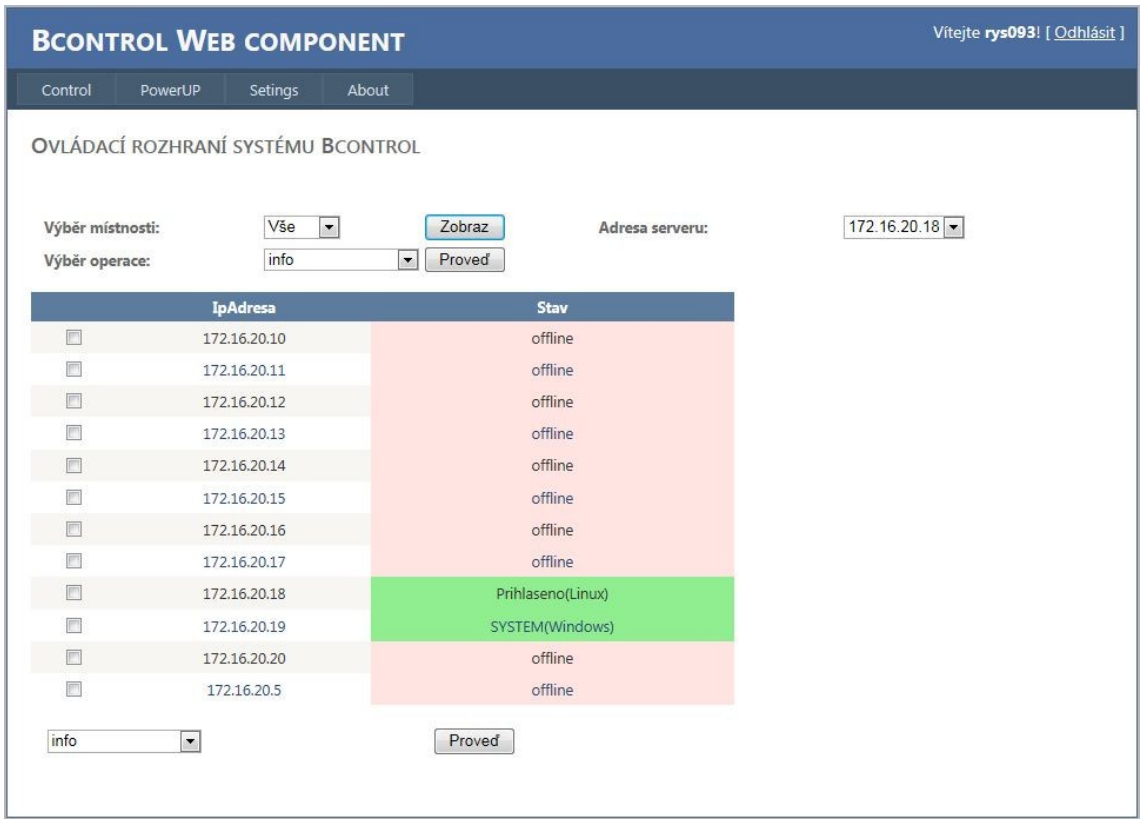
Tato poslední část práce bude věnována tvorbě webové služby, která bude komunikovat se serverem systému Bcontrol a umožňovat pohodlné ovládání veškerých funkcí systému v grafickém rozhraní.

8.1 Použité prostředí a programovací jazyk

Pro tvorbu této služby byl zvolen jazyk ASP.NET a jako vývojové prostředí bylo použito Microsoft Visual Studio 2010. Pro tvorbu byla použita čtvrtá verze .NET Framework knihoven.

8.2 Náhled na uživatelské rozhraní

Uživatelské rozhraní bylo konzultováno s panem Robertem Timkem, který byl správcem sítě Vysoké Školy Báňské: TUO. Veškeré jeho návrhy a připomínky byly zakomponovány do výsledného vzhledu rozhraní, tak aby mu co nejvíce usnadňovalo práci.



Obr. č. 6 Náhled grafického rozhraní

8.3 Programová část

Základem webové komponenty je třída communication.cs, ve které jsou obsaženy metody pro OpenSSL i nešifrovanou komunikaci se serverem.

Nejprve se budu věnovat popisu OpenSSL komunikace a potom v krátkosti popíši i komunikaci nešifrovanou a způsob jakým jsou zpracovávána přijatá data od serveru. Jak již bylo napsáno v části věnované protokolu, server očekává data ve tvaru: „operace|ip adresa|ip adresa…“ a na tento řetězec odešle nazpět buď „ip adresa|stav|ip adresa|stav…“, nebo „ip adresa|prijato|ip adresa|prijato…“, tedy zpracování tohoto řetězce je poměrně snadné.

8.3.1 OpenSSL komunikace

Realizace OpenSSL komunikace byla mírně náročnější, protože ASP.NET tyto knihovny nepodporuje, bylo tedy nutné vytvořit statickou třídu API, kde byly potřebné OpenSSL metody definovány a také přes tuto třídu volány.

```
[DllImport(libssl)]
extern public static int SSL_write(IntPtr ssl, byte[] buf, int num);

[DllImport(libssl)]
extern public static int SSL_set_fd(IntPtr ssl, int fd);
```

Samotná komunikace započne inicializací soketu a zjištění IP adresy z překladače doménových jmen. Následně se pokusí připojit k serveru pomocí klasického soketového spojení.

```
m_socket.Connect(IPs, port);
```

Pokud se spojení podaří, dojde k inicializaci OpenSSL knihoven a postupně se inicializují OpenSSL struktury. Na této ukázce je volba metody OpenSSL komunikace.

```
method = API.SSLv3_client_method();
```

Následuje blok try-catch, ve kterém je realizován samotný SSL_connect a ověřen certifikát serveru. Pokud jakákoli operace neuspěje, dojde k vyčištění datových struktur a běh skončí výjimkou. K tomuto stavu došlo jen při testování z důvodu špatně importované OpenSSL metody. Běžně se vyskytující „chyby“ jak například neběžící server, nebo špatně odeslaná data jsou ošetřena výpisem přímo na zobrazované stránce.

Dále dojde k odeslání dat standardní metodou SSL_write a ověření úspěchu operace, následující operací je SSL_read, která načte odpověď od serveru. Tato data je nutné převést do

odpovídajícího formátu, aby mohla být dále zpracována a použita pro zobrazení v prvku GridView.

```
System.Text.Encoding enc = System.Text.Encoding.ASCII;
string myString = enc.GetString(readBuf);
```

Následné zpracování těchto dat je provedeno metodou parse(), která nahradí veškeré výskyty ukončovacích znaků „\0“ a následně odstraněn případný výskyt znaku „|“ na konci řetězce metodou TrimEnd('|');

```
data = data.Replace("\0", "");
```

Takto upravený řetězec je rozsekán do pole datového typu string, které je následně vloženo do listu dataList, vlastního typu Computer – kde jsou dvojice dat IP adresa a stav.

```
if (poleDat.Length >= 2)
{
   for (int i = 0; i < poleDat.Length; i++)
   {
    dataList.Add(new Computer(poleDat[i], poleDat[i + 1]));
    i++;
   }
}
```

Po této metodě následuje nastavení zdroje dat prvku GridView na naplněný dataList a poté dojde metodou Bold(), k barevnému odlišení sloupce stavu počítače – jak je vidět na obrázku č. 6.

```
foreach (GridViewRow row in GridView1.Rows)
{
    if (GridView1.Rows[i].Cells[2].Text != "offline" &&
    GridView1.Rows[i].Cells[2].Text != "Errr")
   {
           GridView1.Rows[i].Cells[2].BackColor =
           System.Drawing.Color.LightGreen; }
   else { GridView1.Rows[i].Cells[2].BackColor =
    System.Drawing.Color.MistyRose; }
           i++;
    }
}
```

Takto popsaným způsobem probíhá zpracování přijatých dat od serveru – samotná příprava dat na odeslání je pouze složení řetězce z názvu operace zvolené v prvku DropDownList a následně vložené ip adresy počítačů oddělené znakem pípa. Z takto složeného řetězce jsou následně odstraněny veškeré mezery a výsledný řetězec je převeden na pole byte[], které je použito v metodě SSL_write().

```
byte[] buf = Encoding.UTF8.GetBytes(retezec);
```

Příprava dat i odeslání za pomocí OpenSSL komunikace byly popsány, a nyní bude popsána možnost zvolit jen určité počítače z výpisu a zaslat jim určitou operaci. Tato operace byla

provedena přidáním prvku CheckBox do GridView a následně jeho procházení níže uvedenou metodou.

```
foreach (GridViewRow row in GridView1.Rows)
{
   CheckBox ck = (CheckBox)row.Cells[0].FindControl("RowCheckBox");
if (ck.Checked)
{
    retezec += GridView1.Rows[i].Cells[1].Text;
    retezec += "|";
}
```

Takto budou odeslány serveru jen zvolené ip adresy počítačů a odpovídající operace. Tato funkce je dostupná v dolní části stránky, kde je další DropDownList s názvem operace a tlačítko „Proved“.

8.3.2 Nezabezpečená komunikace

Tato komunikace je v programu reprezentovaná klasickými metodami pro soketovou komunikaci. Začátek této metody je shodný s OpenSSL verzí, kdy se webová služba pokouší navázat spojení se serverem a pokud nastane případ, že server neběží, nebo je chybně zadaná jeho IP adresa, tak bude vypsána chybová zpráva.

```
IPAddress[] IPs = Dns.GetHostAddresses(server);
m_socket = new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
try
{
   m_socket.Connect(IPs, port);
}
catch { return "Nepodařilo se připojení k serveru|Errr"; }
```

Po úspěšně navázaném spojení se nejprve přijme od serveru šifra, kterou budou následně předávaná data zašifrována. A poté dojde k zašifrování a odeslání dat, která prošla před odesláním shodnou úpravou jako v případě OpenSSL komunikace. Následně po přijetí dat je provedeno jejich dešifrování a jsou převedeny na řetězec opět stejným způsobem jako v případu OpenSSL komunikace.

Veškeré prováděné kroky jsou v případu nešifrované komunikace shodné, jen přibylo šifrování a dešifrování dat interní šifrou a data jsou odesílána i přijímána, přes inicializovaný soket příkazy send a receive.

```
m_socket.Send(buffer);
```

Celá část nešifrované komunikace zatím není aktivní, jelikož nebylo u serverové části dopředu počítáno s možností kompatibility jazyka s jiným než C++, byla pro bitový posun v nešifrované komunikaci využita template třída CODDING, kterou se nepodařilo v jazyce C# adekvátně nahradit. Nicméně nešifrovaná komunikace je plně funkční, a jakmile se změní

metoda bitového posunu dat v serverové verzi, bude možné ji využít. Byl otestován přenos dat mezi serverem a webovou aplikací za použití nešifrované komunikace a problém se objevil jen při šifrování a dešifrování přenášených řetězců.

8.3.3 Zapínání počítačů

Server umožňuje zapnout vzdáleně počítače funkcí wake-on-lan, odesláním tak zvaného Magického paketu na zvolené MAC adresy počítačů.

Pro tuto operaci je ve webovém rozhraní samostatná záložka PoverUP, která je identická se záložkou Control, jen jsou v prvku GridView jsou zobrazeny MAC adresy počítačů místo jejich IP adres.

Zpracování a odesílání dat je opět identické. U záložky zapínání počítačů se nevolí operace prvkem DropBox jako v případě záložky Control, ale odesílaný řetězec se přímo skládá z klíčového slova wake a MAC adres vybraných, nebo všech počítačů.

Výsledný řetězec potom vypadá následovně:

```
wake|MAC adresa|MAC adresa|…
```

8.4 Možnost konfigurace

S možností výběru IP adresy serveru je zde také možnost manuální změny v databázi uložených adres pro případ změny IP adresy serveru z důvodu jeho přesunutí. Volba více IP adres serveru umožňuje přepínat mezi servery a takto ovládat několik samostatných sítí, kde každá má vlastní server.

Editace je řešená v záložce Settings, kde jsou veškeré IP adresy serverů vypsány v prvku GridView s možností jejich úpravy. Také je zde možno změnit heslo aktuálně přihlášeného uživatele, nebo přidat uživatele zcela nového.

Závěr

Cílem této práce bylo naprogramovat klientskou a webovou ovládací část pro systém Bcontrol, včetně kompletních instalačních balíčků, aby bylo nasazení systému co nejjednodušší. Tato práce poměrně dobře ukazuje, že tvorba systému tohoto rozsahu je velmi náročná na čas i schopnosti programátora.

U tohoto systému jsou kladeny velmi vysoké požadavky na spolehlivost a stabilitu provozu. Proto bylo velmi nežádoucí, aby měl systém nějaké problémy s vlastní stabilitou, spotřebou paměti anebo způsoboval nestabilitu celého operačního systému. Systém Bcontrol byl důkladně testován ve virtuálním prostředí VirtualBox, kde byl zkoušen na verzích 8.04 až 10.10 Linuxové distribuce Ubuntu a také na operačních systémech Microsoft Windows XP a výše včetně 64-bitové verze operačního systému Windows 7. Po testování ve virtuálním prostředí, byl systém Bcontrol v lednu roku 2011 nasazen na asi sto počítačů katedry informatiky a výpočetní techniky Vysoké školy Báňské: Technické univerzity Ostrava, kde se osvědčil jako funkční a byl využíván správci sítě.

Na případu webové aplikace bylo předvedeno použití OpenSSL knihoven na platformě ASP.NET, které u této platformy není nejběžnější, větší mírou jsou využívány klasické SSL knihovny.

Webová ovládací aplikace byla tvořena jako poslední část systému a proto byla otestována pouze ve virtuálním prostředí, kde fungovala bez problémů a tato funkčnost se dá očekávat i u reálného nasazení.

Systém Bcontrol bude dále vyvíjen formou podpory nových operačních systémů a webové ovládací rozhraní bude dále vzhledově optimalizováno dle případných připomínek, které budou vzneseny od jeho uživatelů.

Seznam použité literatury

[1] TROELSEN, Andrew. *Pro C# 2008 and the .NET 3.5 Platform*. Fourth Edition. New York : Apress, 2007. 1333 s. ISBN 978-1-59059-884-9, 1-59059-884-9.

[2] SMITH, Justin. *Inside Windows Communication Foundation*. Redmond : Microsoft Press, 2007. 273 s. ISBN 978-0-7356-2306-4, 0-7356-2306-6.

[3] IBM. *IBM TPB Product Information Center* [online]. 2011. Dostupné z WWW:<http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic=/com.ibm.ztpf-ztpfdf.doc_put.cur/gtpc2/cpp_ssl_get_cipher.html>.

[4] FixUnix. *FixUnix : Forum: Openssl* [online]. 2010. Dostupné z WWW: <http://fixunix.com/openssl/>.

[5] DOSTÁL, Radim . *Root : Seriál Sokety a C/C++* [online]. 2003. Dostupné z WWW: <http://www.root.cz/serialy/sokety-a-cc/>.

[6] The OpenSSL Project. *OpenSSL : OpenSSL Project* [online]. 1999, 2011. Dostupné z WWW: <http://www.openssl.org/>.

[7] HEDENFALK, Martin. *LibConfuse* [online]. 2002, 2010. Dostupné z WWW: <http://www.nongnu.org/confuse/>.

Seznam obrázků